(12)

E-L Users' Manual

Judy G. Townley

February 17, 1994

Software Options, Inc.
22 Hilliard Street
Cambridge, Mass. 02138

This work was supported in part with funds provided by the Defense Advanced Research Projects Agency under contract N00014-85-C-0710 with the Office of Naval Research.

MATERIAL INSPECTION AND RECEIVING REPORT

Form Approved
OMB No. 0704-0248

PLEASE DO NOT RETURN YOUR COMPLETED FORM TO EITHER OF THESE ADDRESSES.

1. PROC. INSTRUMENT IDEN. (CONTRACT)	(ORDER) NO.	6. INVOICE NO./DATE	7. PAGE	OF	8. ACCEPTANCE POINT
N00014-	85-C-0710		1	1	D

3. SHIPMENT NO	4. DATE SHIPPED	4A. BL / TCN	5. DISCOUNT TERMS
SOI-001	23 Feb '94	N/A	N/A

9. PRIME CONTRACTOR CODE	10. ADMINISTERED BY CODE
2V509	S2206A
Software Options, Inc. 22 Hilliard Street Cambridge, MA 02138	DCMAO, Boston 495 Summer Street Boston, MA 02210

11. SHIPPED FROM (If other than 9) CODE / FOB	12. PAYMENT WILL BE MADE BY CODE
	SC1016
	DFAS-Columbus Center ATTN: Bunker Hill Division PO Box 182077 Columbus, Ohio 43218-2077

13. SHIPPED TO CODE	14. MARKED FOR CODE
see Envlosure Number 1 of contract	

15. ITEM NO.	16. STOCK/PART NO. / DESCRIPTION (Indicate number of shipping containers - type of container - container number)	17. QUANTITY SHIP/REC'D *	18. UNIT	19. UNIT PRICE	20. AMOUNT
A0002	Final Report in accordance with A0002 and Exhibit A 15 copies enclosed Distribution made in accordance with Enclosure Number 1 of contract	1	LO	NSP	NSP

21. CONTRACT QUALITY ASSURANCE

A. ORIGIN

☐ CQA ☐ ACCEPTANCE of listed items has been made by me or under my supervision and they conform to contract, except as noted herein or on supporting documents.

DATE / SIGNATURE OF AUTH GOVT REP

TYPED NAME AND OFFICE

B. DESTINATION

☐ CQA ☐ ACCEPTANCE of listed items has been made by me or under my supervision and they conform to contract, except as noted herein or on supporting documents.

DATE / SIGNATURE OF AUTH GOVT REP

TYPED NAME AND TITLE

22. RECEIVER'S USE

Quantities shown in column 17 were received in apparent good condition except as noted

DATE RECEIVED / SIGNATURE OF AUTH GOVT REP

TYPED NAME AND OFFICE

* If quantity received by the Government is the same as quantity shipped, indicate by (✓) mark. If different, enter actual quantity received below quantity shipped and encircle.

23. CONTRACTOR USE ONLY

DD Form 250, DEC 91 — Previous editions are obsolete.

Contents

1 Introduction

E-L is a software development environment that reflects a fresh look at how the design of a programming system and the design of the languages it supports can reinforce each other in ways that increase software productivity. The benefits of this coordinated environment-and-language design[1] are reflected most directly in E-L's tools for manipulating programs. In E-L, one sets up a structure that indicates a desired result. E-L uses a strategy called *opportunistic scheduling* to minimize manual tool invocation and to mediate the objectives of maximizing the use of tools and maximizing responsiveness. Most tool invocations are automatic, typically in response to editing or to actions taken by other tools, which are themselves automatically invoked. Because tools are "scheduled", not run immediately, the system can optimize the use of resources. For example, the system may locate idle machines on the network and use them for some invocations. The system will usually not run more than one scheduled tool on any one machine at a time (to minimize swapping and thrashing).

E-L also supports multiple distributed users in an unusual way. Rather than merely trying to keep different users from harming one another, the goal is to encourage close cooperation. For example, in addition to supporting branching and merging lines of development, E-L also aids users in concurrently editing different but closely related pieces of the same program or document and coordinating the construction of the next version. In particular, a user examining a part of the E-L repository will see changes to that part as they are made, even if by other users or by background tool invocations, perhaps on other machines.

1.1 Artifacts

In most environments one deals with files; in E-L, one deals with *artifacts*, which differ from files in several respects. Perhaps the most significant is that while one can "change the contents" of a file, the contents of an established artifact are immutable. In E-L, when one "edits an artifact", what actually happens is that one creates a new artifact, starting from the existing artifact. The image is that of producing a new edition of a book, rather than modifying what is written on a piece of paper. The latter is the model provided by file systems. The former takes a while to internalize but provides a better basis for tracking the evolution of a constantly changing system. Because an artifact is immutable, it is also a *version*, that is, it is unchanging data created at a particular time. A user of E-L need not resort to an external system like RCS to track versions—it happens automatically as part of using the system.

Another difference between artifacts and files is that an artifact has a *type*. Typing of files is done by some operating systems (e.g., text vs binary), but typing of artifacts in E-L plays a central role in integrating tools. The set of types grows as new tools are added to the environment. In classical file systems, naming conventions are often used to achieve something of the effect of types, but there is generally no enforced correspondence between a naming convention and the contents of a file. Types of artifacts are used in a more classical programming way: they govern the structure of the artifact and the operations that can be

[1] The name E-L is intended to suggest the simultaneous concern with environment and language.

applied to it.

Artifacts can *reference* other artifacts. A reference is more than just a relation between artifacts—a reference emanates from a particular place in the contents of the referencing artifact. For example, an artifact that describes a document might have references to artifacts that describe the chapters, one of which might, in turn, have a reference to an artifact that provides a drawing to be inserted in the document. Unlike file systems, E-L understands these references. For example, you cannot delete an artifact that has references to it. Combined with the immutability, the guarantee of "no dangling references" provides a conceptually simple basis for configuration management. In E-L, a *configuration* is defined to be the transitive closure of an artifact under the references relation. It is the smallest structure containing a given artifact, called the "root" of the configuration, and all artifacts referenced by an artifact in the structure. The rule about not deleting an artifact having references to it means that the integrity of configurations is guaranteed.

The integration of tools involves both the notion of a reference and that of a *derivative*. A derivative of an artifact is an object that can be computed from the configuration rooted at that artifact. A derivative does *not* depend upon any other information than what is in the configuration, such as the time of day or the contents of a setup file. Furthermore, a derivative is associated with the artifact at which the configuration is rooted. Thus, the user does not stumble across an executable and wonder where it came from. Instead, the user points to the artifact that specifies the executable—that is, the one with the executable as a derivative—and says "execute". E-L then finds the derivative and executes that program.

A deriver—that is, a tool that computes a derivative—does not care about the type of a referenced artifact, only about the *kinds* of its derivatives. This is the basis for an open architecture. One can write a new deriver, whose input is a new type of artifact and whose output is an existing kind of derivative. The new tool is automatically integrated: artifacts that reference artifacts of the new type will automatically trigger the running of the new tool when their their tools need a derivative of the existing kind.

1.2 Updating

When several people are working together on a single program, making changes to it requires coordination. Some of the difficulties are unavoidable—everyone wants the system they are using to incorporate all the latest changes, but no one wants the latest bugs that are introduced in the process of making those changes. E-L is not magic, but it does provide some help in allowing several people to work on a program at once, in a reasonably controlled way. You have enough flexibility to step on each other's toes, and enough information to find out who did it.

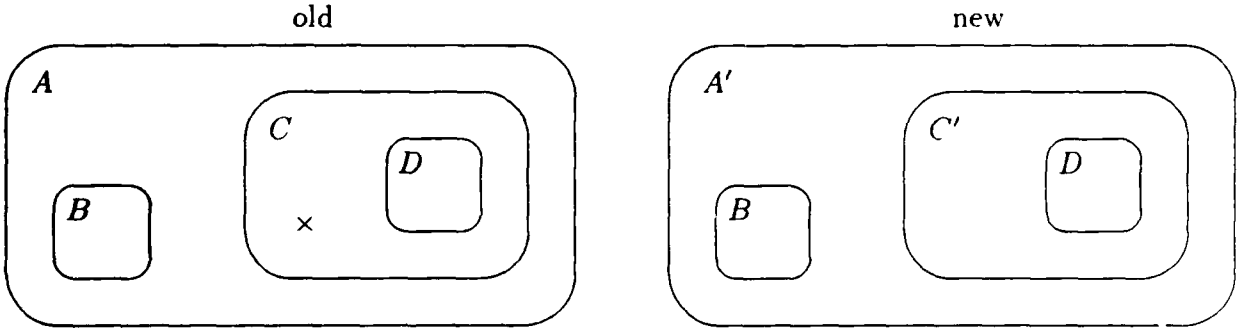
In E-L, when one edits an artifact, what actually happens is that one creates a new *draft*—starting from the existing artifact. The draft comes into existence when editing begins, and it is *committed*, and turns into a new artifact, when editing ends.

There is the question of whether an artifact is *up-to-date*. This attribute of an artifact cannot be part of its contents, because it can change long after an artifact is committed. Artifacts become out-of-date when they are superseded by other artifacts; committing an edit is a common way that this happens. With the notion of "up-to-date", we can correct a

white lie of the previous section:

- Editing an artifact has the effect of editing all artifacts that reference the edited artifact *and that are also up-to-date.*

A visual image is given in the following picture, in which A, B, C and D are the artifacts, and $\times$ marks the change.



After the change, C is out-of-date (and its successor is C') because it was edited directly, and A is out-of-date (and its successor is A') because it references C. Note that B and D do not have successors but are referenced by two artifacts: B by A and A', D by C and C'. The rule that editing affects only up-to-date referencers means that a new version of B will bring forth a new version of A', but not of A. Additional details about the effects of committing a draft are in sections 3.1 and 5.3.

One might worry that propagating the effects of editing would consume tremendous amounts of storage, but this is not so. Clever representation reduces the storage required for both A and A' in the above example to about what it is for one of them alone. Treating them as two artifacts simplifies the bookkeeping for version control and provides a simple mental model to account for the effects of an edit. It also provides a basis for building incremental tools: the derivative associated with a predecessor can be used to compute the derivative of its successor with confidence; one always knows what went into producing the derivative. (Much more can be said about incremental tools than is appropriate in a Users' Manual.)

1.3 Plexes

Up to this point, we have been concentrating on artifacts, whose contents are static. We use the term ***plex*** to describe a portion of the E-L repository whose contents are dynamic. The notion generalizes that of "relation" as it is used in database terminology. A relation is a plex whose state at any one time can be described by a set of tuples, all of the same length. Plexes generalize this by allowing the state to take other forms, for example a list or a tree. (It is understood that relations are logically sufficient; the reason for plexes is that sets of tuples may not be convenient for some purposes.)

A user views a plex with an editor and, to the extent allowed by the plex, uses an editor to change it. Because plexes may be large, viewing is controlled by a *filter* to limit the amount of the plex that the user sees on the screen. While a filter controls which artifacts

a user sees, a *format* controls how those artifacts are presented on the screen. The details of what a user sees when viewing a plex through a particular filter, arranged according to a particular format, are of course dependent upon that plex, as are the specifics of how to specify a filter or a format for the plex. However, the notions of filter and format apply when viewing any plex. The commands for specifying filters and formats are described in section 5.

Several important plexes are built into the system. One of these is the `artifacts` plex, which not only keeps track of all the artifacts in the system, but also various attributes attached to them; these include type, time of creation, whether the artifact is up-to-date, name, and the person who created it. The `artifacts` plex is the means by which a person locates a particular artifact; for example, users generally organize their data so that there is no more than one element in the set of artifacts that is up-to-date and has a particular name and type, or name and creator. The interface for the E-L system makes locating artifacts in this way no more difficult than specifying a file name. It is also common to use the viewing machinery to locate artifacts in other ways. For example, a user may wish to see all artifacts with a given type that are up-to-date and created before a given time or all artifacts with a given name, regardless of their type or whether they are up-to-date. Such subsets of artifacts may be viewed by specifying the appropriate filter. Similarly, a format may indicate sorting first by name or by time of creation or may say to include or omit the display of certain attributes.

In the normal daily life of a programmer, artifacts may be created by the score; a large program may have thousands of up-to-date artifacts. In contrast, the creation of a new plex is a relatively rare event; doing so has more of the flavor of extending the environment than developing a program.

2 Getting Started

The current user interface to E-L, which is not the one of our dreams, is based on EPOCH, an extension of GNU EMACS that runs exclusively under the X window system. So before starting EPOCH E-L, you should be at an X terminal or a workstation running X, talking to a shell that has its `DISPLAY` environment variable set appropriately.[1] Then the shell command `e-l` starts an EPOCH that has been augmented with commands for using E-L.

This manual assumes familiarity with EPOCH, which provides highlighting and multiple fonts, versatile mouse commands, and the use of multiple X windows (called *screens* in EPOCH) in a single EMACS session. EMACS has complete on-line documentation for EMACS and EPOCH, including a tutorial under the command `M-x help-with-tutorial`.[2] All key bindings for E-L commands begin with `C-z`.[3] Each command also has a name, of course, and can be invoked from EPOCH by first typing `M-x` and then the command. You may use E-L to edit files in the usual way, but this will not be your normal activity. To exit, type `C-x C-c`.

Because E-L makes it especially easy to program by writing a document (in which the code is sprinkled), we will introduce E-L by describing how to prepare a document, using LaTeX. We assume some knowledge of LaTeX and defer exposing the full generality of most features until later chapters. By the end of this chapter you will be able to create and edit simple LaTeX documents.

2.1 Drafts

When you are editing in E-L, you are working on a *draft*. For the present, you may think of a draft as corresponding to an EMACS buffer. Unlike ordinary uses of EMACS on files, a draft is not connected to some file whose name you know, but exists unto itself. A draft also has a type. When preparing LaTeX documents, the first type of interest is `latex-root`, corresponding to the "root" file for a document.

There are several ways to begin working on a new draft. One of these initializes the draft to be the default for a type:

- `create-draft` *type name* `C-z C-c`

[1] The `DISPLAY` environment variable is described in the on-line X manual. To check its value, say

```
echo $DISPLAY
```

To change it, say, e.g.,

```
setenv DISPLAY myhost:0.0
```

[2] The EMACS notation for commands is the first thing explained in the tutorial. If you aren't familiar with it, press the Escape key, then `x`, then type `help-with-tutorial` and press Return. The Meta key mentioned in EMACS documentation usually exists on X terminals, but is rarely labeled as such. Try keys marked with ◇, Left, Right, Alt, Compose Character, ...

[3] Non-E-L EPOCH also uses `C-z` as a prefix for commands having to do with EPOCH screens. These key bindings are not available in EPOCH E-L, but you can of course bind such commands to other key sequences in your own initialization file (`~/.emacs`).

For example, suppose you wish to prepare the root for a new document. You can use **create-draft**, supplying "**latex-root**" as the *type* argument[4] and, say, "**trial**" as the *name*. After you supply the name, the following will appear on your screen:

```
\documentstyle[12pt]{article}
\begin{document}

\end{document}
```

The cursor will be on the blank line between the \begin{document} and \end{document} commands.

2.2 Simple Commands

You edit a **latex-root** draft in the usual way. To indicate to the system that you have finished editing a draft, you can use the following command:

- **commit-one-draft** *draft* C-z C-s

The command supplies a reasonable default and asks for confirmation. It uses the draft to make an artifact, which, like a draft, has a type but, unlike a draft, is permanent and unchanging. You will note that, while the buffer remains, it changes to read-only, signifying its change from a buffer on a draft to a buffer on an artifact.

Once you commit a **latex-root** draft, you are guaranteed that LaTeX will be run on it.[5] Once the LaTeX run completes, you may preview the results on the screen, print the results, or examine the log that LaTeX produces. Each of these commands may be applied to a **latex-root** and, in each of the cases, if you ask for something before it is ready, you will be told of the difficulty. We complete the description of these commands in section 3.3.

To preview the output, use the first command to invoke a "dvi" previewer and the second to invoke a full PostScript previewer (if you don't know what the difference is, use the first command):

- **preview** *artifact*
- **ps-preview** *artifact*

If you want to print the output, use

- **hardcopy** *artifact switches*

The contents of *switches* depend upon the particular Unix utility used to print **.dvi** files. Your system administrator can give you the specifics for your site.

You can examine the log that LaTeX produces with

[4] E-L allows completion of type names in the same way that EMACS allows completion of command names. A **TAB** following partial input causes the name to be extended as far as possible. A second **TAB** at that point produces a list of acceptable completions.

[5] Unless you commit it with a **C-u** prefix.

- **latex-log** *artifact*

This command sets up a buffer whose contents are the LaTeX log pertaining to the *artifact*. The cursor is positioned at the end of the buffer, making some of the warnings and error messages (if there are any) visible right away. You will see that the buffer on the log contains references to the artifact(s) involved in the run; each reference is prefixed by the word "Insert", "Begin", or "End". The latter pair are used to indicate that processing the referenced artifact generated the several enclosing lines of output in the log. The buffer-dependent commands

- **follow-link** C-z ^

and

- **follow-link-reverse** C-z M-^

allow you to bounce, in this case, from the log to a buffer on a referenced artifact and back, respectively. We do the best we can, given the limitations of LaTeX's error reporting, to position the cursor in the buffer on the artifact near the cause of the warning or error. While in the artifact buffer, typing two C-xs will bounce the cursor to the opposite end of the offending construct.

The following commands, though perhaps less fundamental to the use of LaTeX, give you access to additional information that it produces.

- **latex-aux** *artifact*
- **dvi-file** *artifact file*
- **ps-file** *artifact switches file*

The first displays the **.aux** file, and the latter two allow you to obtain the raw data for **dvi** and PostScript, respectively. The *switches* are extra arguments passed to **dvips**, and the output is put into the *file* argument. These latter commands are useful, for example, when you want to transmit the data to someone who doesn't have E-L.

2.3 Notices

Errors may arise in computing derivatives and, when they do, they will be transmitted back to you as a *notice*. Notices appear in a pop-up screen, with each notice containing your name, an id for the notice (currently a number), and a short description of the reason for the notice. If, after committing the artifact, you get out of E-L before the notice arrives, you will see the pop-up screen when you next start E-L.

Notices can alert you to other events than the discovery of an error. For example, you may want to know when a job, like the running of LaTeX, is complete whether or not an error was discovered. The value of the variable **job-done-notice** controls whether a notice is sent to you upon job completion.

To get more details on a notice and to delete a notice, use the following commands:

- **display-notice** *user id* C-z C-n

- **delete-notice** *user id* C-z M-n

An undeleted notice is carried over from session to session, so you will eventually want to use the second of these commands. A natural response to the arrival of a notice is C-z C-n to display it, followed by a C-z M-n to delete it. The notices buffer is actually a view of the **notices** plex, about which you can learn more in section 5. The first command, in the case of LaTeX artifacts, calls a generic function

- **examine-problems** *artifact*

that you can also call directly. It will display a buffer on the log file LaTeX created, thus having the same effect as calling **latex-log** on the **latex-root** artifact.

Because we take care to rerun LaTeX automatically as many times, but only as many times, as necessary to catch forward references (of labels, for example), anything that LaTeX reports as a warning is an unresolvable error. Thus we treat these situations, plus conventional errors (in which LaTeX waits for a response from the user), as error situations and generate a notice. You can customize the definition of an error, however, and thereby customize the situations in which the system generates a notice. We describe these features in section 3.4.

2.4 Subsequent Editing

Suppose you have examined the output of LaTeX, as described above, but you don't like the results. We have said that you can't change an artifact. This is because it serves as a version. However, you can create a new artifact, based on the contents of an old one. For this you use

- **edit-artifact** *artifact* C-z C-e

A draft that is started by an **edit-artifact** command may be edited and committed in the same way as a draft resulting from **create-draft**.

3 More About LaTeX Artifacts

As a gentle introduction to E-L, we told you about some of the features for creating LaTeX documents. What follows here is the next chapter in that story.

3.1 References to Other Artifacts

In ordinary LaTeX, you can embed one file in another with an `\input` command. In an artifact, you use a *reference*—where the `\input{`*file*`}` would appear—and omit the `\input` command. A *reference* is not an ordinary sequence of characters. Rather, you must insert it, while editing a draft, with one of several commands.

The first of these that you are likely to want is

- **`create-draft-and-reference`** *type name* `C-z M-c`

Like **`create-draft`**, this sets up a new draft with a given *type* and *name*. In addition, it inserts a reference to that draft at the cursor. The *type* of artifact that most closely corresponds to a file that you would "input" in a LaTeX document is **`latex-piece`**. Suppose that you choose **`latex-piece`** as the *type* and **`trial1`** as the *name*. The text that will appear as the reference is **`[trial1 latex-piece]`**. However, this is not ordinary text, as indicated by the fact that it is highlighted. The characters that appear are just a readable way of displaying what is in reality a reference to another draft. To edit the draft, simply switch to the buffer labeled **`trial1 latex-piece`**, using ordinary EMACS commands like `C-x b` or `C-x C-b`.

You can also insert a reference to an existing artifact or draft into a draft buffer (at the position of the cursor) with the following commands. The first two commands are most appropriate when you want to identity the artifact or draft to insert by its name.

- **`insert-artifact-into-draft`** *name*
- **`insert-draft-into-draft`** *name*

The following command uses the cursor position to identify a default argument. That is, if your cursor is near the artifact or draft you wish to insert, you can use

- **`copy-as-kill`** `C-z x`

This command copies an artifact or draft reference into the kill-ring, from which you can yank it (into the draft buffer, for example) using an ordinary EMACS command like `C-y`.

When you have finished editing the reference, you can use the **`commit-one-draft`** command to commit it. One aspect of committing that you must be aware of from the start is the following: you may not commit a draft that references other drafts. Continuing the previous example, suppose that you first try to commit the **`latex-root`** draft named **`trial`**, before committing the **`latex-piece`** draft named **`trial1`**. You will see the following message in the minibuffer:

```
This draft references a draft
```

You may commit the drafts one by one bottom up (i.e., `trial1` first) or use one of the commands described in section 4.3 that commit several drafts at once. In all cases, when you commit a referenced draft, the reference changes to include a *timestamp*, indicating the date and time of committing the draft. (We say more about timestamps in section 5.2).

Returning to the specifics of LaTeX artifacts, the contents of a `latex-piece` artifact is ordinary input to LaTeX, except for its references to other artifacts. The semantics of a reference from either a `latex-root` artifact or a `latex-piece` artifact to a `latex-piece` artifact is essentially that of textual inclusion. However, a `latex-root` or `latex-piece` artifact can reference other types of artifacts as well. For example, if E-L supports programming in a given language, it will support typesetting of program fragments in that language. A reference from a `latex-root` or `latex-piece` artifact to an artifact whose type indicates that it holds source text for the language will result in a typeset version of the program fragment appearing at the point of reference.

The usual rule is that, when a `latex-root` or `latex-piece` artifact references an artifact, the `manuscript` derivative of the reference is inserted. Each artifact type has its own rule for how `manuscript` derivatives for artifacts of that type are produced, if at all. There is enough flexibility here that no general rule is possible. For example, the `manuscript` derivative of a `latex-piece` artifact is simply the contents of the artifact, with suitable provisions for obtaining the `manuscript` derivatives of its references. On the other hand, a `latex-root` artifact has no `manuscript` derivative—what with the `\documentstyle` and `\begin{document}` and `\end{document}` commands, a reference from another document to such text would only confuse LaTeX. To learn about the `manuscript` derivative for a specific type, you have to read about the details for that type.

An exception to the usual rule occurs when a reference to an artifact is immediately followed by `<caption>`. In this case, the `caption` derivative, rather than the `manuscript` derivative, is inserted at the point of reference. Like `manuscript` derivatives, each type has its own rule for producing a `caption` derivative, but, as the name suggests, `caption` derivatives are intended to be short. The `caption` derivative of a `latex-piece` is seldom used; consequently, the default simply indicates that the user did not provide a caption. To specify a non-default caption, start the first line of the artifact with `%%Caption:` the remainder of the line following the : and stripped of leading and trailing whitespace, will be used for the `caption` derivative. You can specify a `caption` derivative for a `latex-root` similarly, useful when referencing a `latex-root`.

3.2 Creating an Index or Bibliography

To create a document with an index, you need to do two things: put a `\makeindex` command in the preamble before the `\begin{document}` command in the `latex-root` (just as you would do in ordinary LaTeX) and put a `\printindex` command where you want the index to appear (where you would put `\input{`*file*`.ind}` in ordinary LaTeX).

You can engage BibTeX's help in producing a bibliography by using a `latex-bib` artifact type. The text you put in a `latex-bib` artifact is the same that you would put in a `.bib` file. A `latex-root` artifact may reference one or more `latex-bib` artifacts; each should appear just before the `\begin{document}` command—that is, after any `\makeindex` command or

references to other **latex-piece** artifacts—with **<bibliography>** immediately following the reference. The derivative of a **latex-bib** artifact is another exception to the rule that references in **latex-root** or **latex-piece** artifacts supply **manuscript** derivatives. Its derivative is a **bibliography**, which is used like a file argument to the **\bibdata** command. Again, where you want the bibliography to appear, use, in this case, a **\printbibliography** command (in the same way you would use the **\bibliography** command in ordinary LaTeX)—along with a **\bibliographystyle** command, if desired. In combination with the usual **\cite** command, these commands and the use of **latex-bib** references will produce citations and a bibliography in the document. E-L does all the delicate, interleaved invocations of LaTeX and BibTeX automatically, something that you won't appreciate if you haven't struggled through it.

As a matter of style, one typically puts references in the **latex-root** to **latex-pieces** containing the **\printindex** or **\printbibliography** commands. You can examine a log that describes the results of the **\printindex** or **\printbibliography** command.

- **makeindex-log** *artifact*
- **bibtex-log** *artifact*

These commands work like the **latex-log** command, setting up a buffer whose contents are the appropriate log. The cursor is positioned at the end of the buffer, making some of the warnings and error messages (if there are any) visible right away.

A single command

- **latex-summary-file** *artifact name*

displays any of the many log and summary files that the various processors of a manuscript produce. The first argument must be either a **latex-root** or **latex-piece** artifact, and the second argument is the name of the summary file that you want to see. (You can see the list of possible file arguments in the usual way, by asking for completion with a [t], after supplying the *artifact* argument.)

3.3 Dividing a Document into Parts

In using ordinary LaTeX for a large document, the **\include** and **\includeonly** commands divide the document into smaller parts on which LaTeX can be run. To achieve the same effect with artifacts, place a **\parts** command immediately after the **\begin{document}** command. Each reference is then treated as a separate part and E-L will rerun LaTeX only on those parts affected by an edit.

If you apply one of the commands mentioned in section 2.4 to a **latex-root** with parts, you will be prompted for the name of a part. You can also supply just the name of the part if it is referenced by only one **latex-root**. As this suggests, the full story on the commands is

- **preview** *artifact* [*part*]

which assumes it will not encounter any PostScript in the artifacts;

- **ps-preview** *artifact* [*part*]

which is for previewing artifacts that contain PostScript; and

- **hardcopy** *artifact* [*part*] *switches*
- **latex-log** *artifact* [*part*]
- **latex-aux** *artifact* [*part*]
- **dvi-file** *artifact* [*part*] *file*
- **ps-file** *artifact* [*part*] *switches file*

The **latex-log**, **latex-aux**, **dvi-file**, **ps-file**, and the preview commands *require* a part if you have used the **\parts** command. The **hardcopy** command will print either a **latex-root** or a part.[1] If you supply a **latex-root** artifact as the argument to **hardcopy**, the system asks for a part name, assuming you typically want to print part of a document. You can reply with an artifact or type a carriage-return. If the latter is supplied, the system will ask for confirmation that you want to print the whole document. If you type "no", it will prompt you again for the name of a part; if you type "yes", it will print the whole document.

The **examine-problems** command similarly takes advantage of the finer structure of a **latex-root** artifact with parts. A document with parts typically has problems with more than one part; the **examine-problems** command will present you with a list of just these parts, from which you can use **follow-link** to see the **latex-log**s. If only one part has an error, the **examine-problems** command will take you directly to the log for that part (rather than presenting you with a one-line list).

If you ask for something to be done for a part when a part is expected, no second argument is necessary if the part is referenced in exactly one up-to-date **latex-root** artifact. (We say more about what *up-to-date* means in section 4.3.) If you apply a command to a part referenced by more than one up-to-date **latex-root** artifact, you will see the error

```
Sorry - can't deal with more than one referencer yet
```

If there is no **latex-root** artifact referencer of the given part, you will see the error

Artifact *name* **has no referencers**

To summarize, if you use **\parts**, your **latex-root** artifact would conform, in general, to the following structure

```
\documentstyle[12pt]{article}
preamble
[artifact]<bibliography>
⋮
```

[1]The utilities used for printing and previewing depends upon the user's installation. The values of the variables **hardcopy-command**, **preview-command**, and **ps-preview-command** (which can be examined in EMACS with **M-x describe-variable**) indicate which will be used.

```
[artifact]<bibliography>
\begin{document}
\parts
[artifact]
⋮
[artifact]
\appendix
[artifact]
⋮
[artifact]
\end{document}
```

where anything that you want in the *preamble* could be contained in references. You can count on the following to guide your use of LaTeX artifacts.

- If you change anything in the *preamble*, LaTeX will be rerun on everything.
- If you change, add, or delete a bibliography reference, LaTeX will be rerun only on the changed references and those parts whose citations are affected.
- If the document uses **\parts**, you can change a reference and/or add and/or delete references after the **\parts** command and E-L guarantees LaTeX is rerun on every part and only those parts affected by a change. For example, if you add a new reference that changes the section or page numbering in the rest of the document, LaTeX will be rerun on subsequent sections. On the other hand, if page numbers are start at 1 at the beginning of each part, then changing the number of pages in one part will not cause LaTeX to be rerun on subsequent parts.
- E-L will rerun LaTeX on parts affected by a change until the results (such as labels and counters) stabilize—that is, until subsequent runs would leave the output unchanged.

If the document does not use **\parts**, LaTeX is always run on the entire document, so the issue of where it must be rerun does not arise.

3.4 Customizing Errors

After running LaTeX, we scan several log files to look for errors. For this scan, we use three regular expressions (re's). One is for the LaTeX log (log), one for the BibTeX log (blg) and one for the makeindex log (ilg). You can substitute your own re's for the defaults. To do so, you add comments of a particular form to the **latex-root**. For instance, if you included

```
%watch-log: "^! \\|^LaTeX \\(error\\.\\|Warning:\\) "
%watch-blg: "^Warning--\\|^(There w\\(ere\\|as\\) [0-9]+ error messages?)$"
%watch-ilg: "done ([^)]*[1-9][0-9]* rejected)\\."
```

anywhere in your root artifact, you would preseve the default behavior. Note that the re's themselves must be double-quoted string constants. They obey EMACS Lisp escape conventions, so, for example, the constants "\n", "\C-j", and "\012" are equivalent, and each contains a single character, i.e., newline. There can be spaces and tabs between the **%watch-xxx:** and the string constant. Anything at all can follow the string constant, e.g., some explanatory words.

Because these regular expressions must be skipped entirely by LaTeX, it's not feasible to let quoted re's cross line boundaries. So none of the string constants can contain an actual newline character, although any of the escape sequences mentioned above are permissible. What's not allowed is something like

```
%watch-ilg: "[0-9]+[^
]*errors"
```

because LaTeX will try to interpret each line that doesn't start with a percent sign. The equivalent

```
%watch-ilg: "[0-9]+[^\n]*errors"
```

would be fine. To build up a long re, you can put several **%watch-xxx:** lines together, with the same **xxx** in each. For example

```
%watch-log: "^! \\|^LaTeX \\(error\\.\\|Warning:\\) "  (Default error checks)
%watch-log: "\\|^\\(Over\\|Under\\)full \\\\[hv]box "  (Knuth aesthetics)
```

produces a single re for scanning `.log` files. It is obtained by concatenating the two string constants. Note that the whitespace and comments at the ends of the lines are ignored. The lines for a given **watch-xxx** declaration *must* be contiguous, and there can be at most one such declaration (possibly occupying multiple lines) in the **latex-root** for each of the three expression types. Furthermore, the (concatenated) re must be a valid EMACS regular expression. If any of these rules is broken at commit time, an error will be raised to prevent commitment.

3.5 postscript Artifacts

This section is of interest to aficionados of LaTeX and PostScript who want to use PostScript in documents, for which we offer a **postscript** artifact type. A **postscript** reference can appear anywhere that a **manuscript** derivative is expected. When you create a **postscript** artifact, the first line must contain a *macro string* that can serve as the replacement text of a TeX macro that conforms to the following rules:

- It may not have [0] (the null character), [1] (**C-a**), or [n](newline).
- It must have a single parameter, denoted in the usual way: **####1**.
- In the eventual expansion of the macro string, one of the following macros must be applied to the parameter:

- oo **\ELPH**—insert the PostScript file in the "header", where it will be outside the save or restore context of individual pages. This is typically used for setting up PostScript libraries.
- oo **\ELPP**—insert the PostScript file where it will remain only for the current page. This is typically used for insertion of PostScript pictures.

In other words, the first line of a **postscript** artifact might be

```
Macro string: \ELPP{####1}
```

which, in fact, is the default in an initial **postscript** draft buffer. The ultimate effect of a reference to a **postscript** artifact is that the text of its contents, after the first line, is made into the body of a PostScript function that is called at the point of reference.

4 More About Artifacts and Drafts

4.1 As Arguments to Commands

The default for an argument identified in this manual as *artifact* is taken from the cursor if possible. If the cursor is on a line that references an artifact, the artifact just after the cursor or else the last artifact on the line becomes the default argument. If the cursor is in a buffer on an artifact, but not on a line referencing another artifact, the artifact becomes the default argument. If you wish to refer to an artifact other than the default, simply type its name. You will be prompted for a type when one is needed for disambiguation. Yet other ways to use the cursor to identify an artifact argument are mentioned in section 5.2. The conventions for specifying a draft argument are analogous.

Once an artifact argument appears in the minibuffer, with the cursor at the beginning of the buffer, you can navigate to neighboring artifacts. [Ephemeral navigation is not yet available for drafts.] A neighbor is chosen with an EMACS-like command:

- **C-p**—referencers (previous/up)
- **C-b**—predecessors (back)
- **C-f**—successors (forward)
- **C-n**—references (next/down)
- **DEL**—previous default (where you just came from)

The **DEL** command is particularly handy if you have navigated from artifact A to one if its referencers, B, and then want to get back to A without going through a menu of B's references. Just hit **DEL** and you'll be back at A. If there is a unique neighbor in the direction indicated, it immediately becomes the new default without an intervening menu.

The navigation menu is an EMACS buffer invoked in a mode similar to the one used by electric-buffer-list (**C-x C-b**). It lists the artifacts one step away from a given artifact in the direction mentioned the menu's first line. The order of lines and the format of each line is the same as the default format on the **artifacts** plex. To select a new default artifact, place the cursor on the appropriate line and type a space. To get more information about menu keys, type your help key followed by "m" while in the menu (i.e. get help on the menu buffer's mode).

Some commands require two artifact/draft arguments. These commands will take their first argument from the mark, if possible, and their second argument from the cursor, if possible. Sometimes you want to cut and paste references into a draft buffer. For this, and for multi-artifact/draft commands, you can use the kill ring as you would in ordinary EMACS. That is, you can put references into the kill ring with the usual functions (**C-w**, **M-w**, **C-k**, **M-d**, or **C-d**, with or without a numeric argument), and you can yank them from the kill ring with the usual functions (**C-y** or **M-y**). This means you can cut and paste text within a draft buffer or even between draft buffers, using ordinary edit commands and not lose artifact references. You can also use the command **copy-as-kill** mentioned in section 3.1 to copy a reference into the kill ring.

4.2 Further Dealings with Artifacts

To examine an artifact without creating a draft, use

- **examine-artifact** *artifact* C-z C-x

This creates a buffer on the artifact, exactly like the buffer obtained after committing a draft. To examine an artifact in another window, use

- **pop-to-artifact** *artifact*

It is sometimes useful to base an edit on several artifacts, rather than zero (**create-draft**) or one (**edit-artifact**). The following command allows you to do this.

- **append-artifact-to-draft** *artifact draft*
  - The draft must not be owned by another session. If not owned by this session, it is taken over.
  - The artifact and draft may be of different types, but, if they are too dissimilar, you will get an error message. The command attempts to put the pieces of the artifact into reasonable places in the draft.
  - The artifact becomes the predecessor of the draft.

You can append several artifacts to the draft; each becomes a predecessor of it.

As you continue editing, you will generate more and more artifacts. Rather than rely completely on users to delete irrelevant artifacts, the system includes a customizable automatic deletion mechanism, described in appendix B. Situations also arise in which you do want to delete one or more artifacts directly. An artifact may be deleted only under the following conditions:

- The artifact is not referenced by another artifact.
- There is no active draft based on the artifact.
- There is no other process (such as a user) that has it cached.

If these conditions are met, you may delete artifacts with the following commands:

- **delete-one-artifact** *artifact* C-z C-d
- **delete-artifact-and-references** *artifact* C-z M-d
- **delete-artifacts-in-region** C-z M-D

The first command confirms the deletion or says why it was not done. The second command attempts to delete the given artifact, and, if this was successful, attempts to delete referenced artifacts, and so on. It reports how many artifacts were deleted (perhaps none), but does not indicate why any particular artifact was not deleted. The third command is usable only when the cursor is in a view on the **artifacts** plex. It deletes all the artifacts between the

mark and the cursor, including the end points. Note the uppercase D in the command, which is there to make it difficult to issue the command accidently. The command announces the number of artifacts it has deleted, it will say when it's done, and it will update the view on the **artifacts** plex as it goes. If an attempt to delete multiple artifacts is unsuccessful, you can, of course, try deleting one of the artifacts (not referenced by any other artifact). If that is unsuccessful, you will learn why. (When you delete an artifact, its predecessors become up-to-date. Thus, even after you delete an artifact, you may still see an up-to-date artifact with the same name, but a different creation date.)

As with any repository system, you may eventually find that you want to export data stored in it—to archive the data off-line or to move it to another, compatible repository—in a form that permits it to be imported at a later date. The first command that you will need is

- **write-archive** *artifact file* [*pred-p*]

which stores the appropriate representation of the *artifact* and all its direct and indirect references in *file*. If the command is prefixed with two or three C-us, it will archive all of the artifact's direct and indirect predecessors as well. The mate of this command is

- **read-archive** *file* [*override-timestamp*] [*override-creator*]

which, by default, will restore the original timestamps and creators of the **archived** artifacts if possible. If the command is prefixed by one or three C-us, the current date and time will be used for the timestamp. If the command is prefixed by two or three C-us, the current user will become the creator. (Section 5.2 describes the timestamp, creator, and other attributes of artifacts.)

Each of the following commands has to do with derivatives. After being asked to confirm the artifact, you will be prompted for the kind of derived information that you desire. The system also provides completion for kinds of derived information, so, by pressing tab at the prompt, you will see your choices. The commands also will take the kind argument, like the artifact argument, from the cursor's position if possible. A reference in a buffer to a derivative, again like an artifact, is not ordinary text, as indicated by the fact that it is highlighted. The first command allows you to examine a derivative of an artifact.

- **examine-derivative** *artifact kind* C-z M-x

You can call **examine-derivative**, for example, with a **latex-root** and an **error%latex-directory** as arguments; this has the same effect as calling **examine-problems** on the **latex-root** (if it has no bad derivatives). It is a generic function that you can apply to any type of artifact, however. You can delete a derivative.

- **delete-derivative** *artifact kind* C-z d

Though the need seldom arises, you can request the derivation of a particular kind of derivative from an artifact.

- **derive-from** *artifact kind*

4.3 Further Dealings with Drafts

We mentioned earlier that there are commands to commit several drafts at once. One of these is the following command, which, if it finds a referenced draft, first attempts to commit it and, recursively, its references.

- **commit-draft-and-references** *draft* C-z M-s

If the draft that you are trying to commit references drafts that someone else is editing, however, the commit will not be successful; you must get the other person to commit first. Another command

- **commit-draft-and-references-and-referencers** *draft* C-z M-C-s

attempts to commit drafts that reference the current draft and those that it references. In fact, it tries to commit *every* draft that it reaches by following reference relationships, in either direction.

To commit a draft without scheduling any tools (such as LaTeX) to be run on its behalf, precede the commit command by C-u, a standard "prefix" argument in EMACS. This is useful, for example, if you expect to make several passes over a part or parts of a document and want to commit them periodically but delay running LaTeX until things settle down.

You may wonder what happens to an up-to-date artifact, like **trial** in the example in section 2, when a referenced artifact, like **trial1**, is edited. Is it up-to-date or out-of-date? The answer is *almost out-of-date*. When the user creates a draft of **trial1**, the system generates a draft of **trial**, call it *d*. Starting a draft doesn't make an up-to-date artifact out-of-date, but E-L does consider an artifact that has a draft successor to be almost out-of-date. The user must explicitly commit *d* to create an up-to-date successor artifact, after which the original **trial** is simply out-of-date. A user may want to edit a number of referenced artifacts before bringing forth an up-to-date referencing artifact. (The command **commit-draft-and-references-and-referencers** is provided for this very purpose.) We state some additional rules linking the notions of up-to-date, almost-out-of-date, and out-of-date in section 5.3. If you commit a draft of, say, **trial1**, you will have two artifacts named **trial1**; if you use the name in a subsequent **edit-artifact**, it will be assumed that you mean the most recent one.

To rename a draft, use

- **rename-draft** *draft name*

To delete a draft, use

- **delete-draft** *draft*

If another draft references the draft you're deleting, the reference will revert to a predecessor. If the deleted draft has no predecessor, the reference to it will become empty.

If there is a crash while you were editing, drafts that you were working on become "abandoned". To continue working on them (from the last autosave, with the cursor restored to the position it had when the draft was abandoned), you may use the command

- **takeover-draft** *draft* `C-z e`

This command can be used to resume working on any "unhosted" draft. See section 5.3 to learn more about system and abandoned drafts, both of which are considered unhosted. You may deliberately abandon a draft with the following command:

- **abandon-draft** *draft* `C-z C-a`

A typical use of this command is in handing a draft off to someone else to finish editing, without having to do an intervening commit. If the system finds any drafts lying around when you exit E-L, you will be given the option of abandoning or deleting them before exiting. You can also `C-g` and pursue other options, such as committing drafts. If you get out of EMACS in some other way, such as by exiting a window system, all your drafts automatically become abandoned.

If you attempt to edit an artifact that has a draft successor, you will be warned that the artifact is "almost-out-of-date" and asked if you wish to edit it anyway. If you respond with a **yes**, and eventually commit both successor drafts, you will end up with two up-to-date artifacts with the same name and type. If you respond with a **no**, you will be asked if you wish to takeover the draft. If you respond **yes** and the draft successor is unhosted, you will simply become the creator of the draft and may proceed to edit it in the normal way. If the draft is hosted, you may edit it only if you are the creator. If you respond **no**, the system will prompt you again for an artifact to edit. You can find the successor draft of an artifact by issuing the command

- **find-successor-draft**

while in a view on the **drafts** plex.

Sometimes an editing mistake will so destroy the contents of a buffer that undo will not restore it properly. If this happens, the following command will restore the buffer from the last autosave

- **revert-buffer** *draft*

with the cursor restored to its position at the time of the autosave.

4.4 Further Dealings with Problems

The command, **examine-problems**, in general, provides information about **bad** or **error** derivatives. The former occurs, for example, when a tool crashes or is aborted; the latter, when a tool signals an error. If the **examine-problems** command finds only one problem derivative—the typical case—it takes you directly to a buffer on that derivative. (In the case of LaTeX artifacts, this is an **error%latex-directory** derivative.) If none exists, it tells you that. If there is more than one problem derivative, it sets up a buffer with one line per problem in it, indicating the **bad** or **error** derivative, a short description of the problem, and a button to follow to learn more about the problem—or * if there is nothing further to examine. Placing your cursor on the button and issuing the **follow-link** command will examine the *kind* derivative of the artifact if the artifact button is followed by <*kind*> or will examine the artifact otherwise. If your cursor is not on a button, but on a line without a *, the **follow-link** command will examine the problem derivative.

4.5 Comparing

The two commands that highlight differences between artifacts use a single command **diff-buffers** that highlights differences between any two EMACS buffers. This command prompts for two argument buffers and, with a prefix argument (**C-u**), asks for a result buffer. The default result buffer is ***merge***. Using the Unix utility **diff**, **diff-buffers**[1] combines the first two buffers and, in the third buffer, displays

- their common parts with the standard background,
- data in the first but not in the second on a red background, and
- data in the second but not in the first on a green background.

The **unbutton** command **C-z u**, when issued with the cursor in a differing (red or green) region, removes the highlighting; with a prefix argument (**C-u**), it deletes the highlighted text. You can move to the next, or previous, differing region with the **next-diff-button-pair** command **C-z >**, or **previous-diff-button-pair** command **C-z <**, respectively. You can even customize the colors that **diff-buffers** uses with the commands **set-diff-button-colors** and **set-diff-reference-colors**; the latter is relevant when using the following artifact-oriented commands.

The command

- **diff-artifacts** *artifact*$_1$ *artifact*$_2$ *buffer*

is similar to **diff-buffers**, except that the first two arguments are artifacts not buffers. The result buffer has not only the red and green regions from **diff-buffers**, but also the usual blue regions displaying artifact references. Artifact references in differing regions still have a blue background, but the letters are red or green indicating that they are in differing regions. The command

- **diff-artifacts-recursively** *artifact*$_1$ *artifact*$_2$ *buffer*

takes the same arguments as **diff-artifacts** but lists in the third argument the pairs of differenced artifacts, one line per pair of artifacts. A **C-z ^** (**follow-link**) in this buffer switches to the buffer where the differences are displayed; killing such a buffer deletes the line. This command will compare artifacts that seem to be in the same place in the two referencing artifacts, without regard for whether they have the same name.

As with other commands that take two artifact arguments, these commands will use mark to offer a default for the first argument and the cursor to offer a default for the second argument. If it is not convenient to have both artifacts in the same buffer, you can put one of the artifacts in the kill ring and, when **diff-artifacts**[**-recursively**] prompts you for the artifact, yank the argument into the minibuffer. (It will be blue in the minibuffer, too.)

[1] The present implementation requires EPOCH and color monitors.

4.6 Searching

To search through a set of artifacts, you can use one of a couple of commands.

- **re-search-artifacts-recursively** *artifact regular-expression filter* [*prefix*] C-z s
  - Starting at the given *artifact*, this command searches for the given *regular expression* in its contents. With no *prefix* argument, it stops when a match is found, at which point you should respond with one of the following keys:
    * **c**—continue the scan
    * **s**—suspend the scan
    * **m**—mark this occurrence by placing it in a log buffer and continue the search
    * **!**—continue the scan without further interaction, marking each match.

    With a *prefix* argument, it does not stop when a match has been found but behaves as if **m** had been typed.
  - After searching the contents, the algorithm is applied recursively to each of the artifact's references that pass the *filter* and have not yet been scanned.
  - The *filter* is accepted in the same way as the filter for a view on the **artifacts** plex. The first time this command is executed the default filter is one that includes all artifacts. For subsequent commands, the default filter is the filter given to the command previously.
  - If the *prefix* was used or if you responded with **m** or **!**, the command switches to a buffer having a line for each occurrence that you marked. You can jump to that occurrence in the usual way, with C-z ^.
  - The **re-search-artifacts-recursively** command names its buffer ***Re-Search Artifacts***. To get two searches going at once, simply rename this buffer and start another search.

- **re-search-artifacts-recursively-continue** [*log-buffer*] [*artifact*] [*prefix*] C-z ,
  - The purpose of this command is to allow you to resume a suspended search or extend a completed search. A suspended search is associated with a buffer. If there is only one such buffer, you will not be asked for the *log-buffer* argument.
  - If the previous search was suspended (using **s**), the *artifact* argument is not requested and the scan resumes just after the last match detected. (It remembers where it was in the recursion.) Previously scanned artifacts are not rescanned.
  - If the previous search exited normally, the *artifact* is requested and a new scan begins at that point. Again, previously scanned artifacts are not rescanned.
  - The *prefix* argument plays the same role as in the previous command—it says to mark matches without interaction.

4.7 Highlighting

Several artifact/draft types have contents that indicate certain information by highlighted regions, where by "highlighted", we mean having a special color in a buffer. This is in addition to the use of highlighting for artifact references and the use of inverse video to indicate the format of contents. The visual effect of the highlighting and, of course, the interpretation depend upon the type. The command that you use to highlight text is independent of type.

Several of the current artifact/draft types, including those that support programming, support two uses of highlighting. You indicate your intent in a particular use of highlighting by giving a numeric argument to the highlighting command.[2] (If you specify a numeric argument that is greater than the number of kinds of highlighting for that buffer, the maximum for that buffer is used instead.) The default value of the numeric argument is 1, for the more common kind of use. Thus, for the common case, you need no argument; for highlighting with a prefix argument of 2, you can use `C-u 2`. You may also use `M-`*n* in all cases.

To highlight text, use

- **highlight-region**[3] *n* `C-z C-h`
  - Using the prefix numeric argument *n*, highlights the region from mark to point. You may see other changes to the buffer, depending upon the type of the draft.
  - If your cursor is in a highlighted region, this command will unhighlight the region.

Instead of mark and point, you can use left-mouse click and drag to indicate the region. If you move your cursor into a highlighted region and begin typing, the new text will become part of the highlighted region.

4.8 Annotating

The artifact type **note** allows you to attach a note to an artifact, much like you might attach a Post-It to a document. The contents of a **note** is unstructured and, in particular, may freely reference other artifacts. A **note** artifact differs from other types of artifacts in several ways: it has no derivatives, the system does *not* automatically create a successor draft to a **note** artifact when you create a successor to an artifact it references, and the system does not automatically delete **note** artifacts. You could use a **note** artifact, for example, to attach a version number and list of recipients to a system artifact (that would naturally continue to evolve) or to associate profiling data with an artifact for future comparisons.

[2] See the section "Numeric Arguments" in the *Gnu Emacs Manual*. In the fifth edition (version 18), this is section 4.9, page 29.

[3] Interactively, the arguments are mark and point and a prefix argument indicating the kind of highlighting, or 1 if none has been given.

5 Plexes

5.1 Introduction

We use the term plex for that portion of the E-L repository that describes dynamic relationships, such as among the artifacts, drafts, and their various properties. Plexes may involve large amounts of information, much of which you are not interested in at any one time. Consequently, a buffer on a plex has an associated filter to limit the amount of data that you see. Filters not only allow you to reduce clutter on the screen, but also result in faster interaction, because EMACS is dealing with smaller buffers. A buffer on a plex additionally has an associated format to control how the information in the plex that passes the filter is presented on the screen. The contents of filters and formats depend upon the particular plex and cannot be described in general. However, the commands for examining the information in plexes allow you to specify filters and formats in a uniform way.

To open a buffer onto a plex, use the following command:

- **view-plex** *plex* [*filter*] [*format*] `C-z C-v`
  - The second and third arguments are optional; a value for *filter* is requested if the command is prefixed by one or three `C-u`s, and a value for *format* is requested if the command is prefixed by two or three `C-u`s. (Thus three `C-u`s means that both a filter and a format are requested.) Default values are used if these arguments are unrequested.
  - This command opens a buffer on *plex* and makes this buffer visible in a window.

A buffer on a plex may be killed in any of the usual ways of killing a buffer.

You may change the filter or format on a buffer by issuing one of the following commands while the cursor is in the buffer. If they are given when the cursor is not in a buffer on a plex, you will get an error.

- **refilter** *filter* `C-z C-i`
- **reformat** *format* `C-z C-o`

A fundamental point to keep in mind is that a buffer opened on a plex is actually a buffer of characters representing the "real" plex. You can think of it as a view onto the plex. Do not confuse the commands that change this view with commands that can actually change a plex.

You will notice that a buffer on a plex is always read-only. This does not mean that you cannot change a plex, but you usually can change plexes only in a very controlled way, with the details depending upon the particular plex.

To find out the names of the available plexes, you can say

- **view-plex plexes**

The default format of the **plexes** plex is an alphabetized display. You may change this plex only by adding or removing plexes. You can get on-line documentation of many of the plexes by issuing the command

- **buffer-documentation** C-z ?

with your cursor in the plex about which you want information.

5.2 The artifacts Plex

The **artifacts** plex is used to view all the artifacts in the repository. It displays artifacts, one per line, permitting the use of the cursor position in the **artifacts** plex to identity an artifact argument to commands. Both filters and formats for this plex are based on *universal attributes*, attributes that exist for every artifact. The current universal attributes are

- **name**—this can be any sequence of inking characters.
- **type**—one of the types recognized by the system.
- **creator**—the user who created the artifact.
- **timestamp**—the time, down to the second, when the artifact was created. These are displayed according to the format

  $$year/mo/da\_hr:mn:sc$$

  In this format, all fields have two digits except *year*, which has four.
- **up-to-date**—a flag indicating whether the artifact has been superseded by one or more artifacts, or whether a referenced artifact has been superseded, and so on, recursively.

The system automatically completes names of attributes for filtering or formatting. A filter on the **artifacts** plex is a set of filters on the individual universal attributes, and an artifact passes such a filter if each of its attributes passes the corresponding filter on the universal attribute. The syntax and semantics of filters on universal attributes are

- **name**—a regular expression.[1] A name passes the filter if the regular expression matches the entire name. (This tends to discourage the use of characters that are special to regular expressions in the names of artifacts, e.g., + and *.) The default is .*, which matches all names.
- **type**—a set of types. A type passes the filter if it is in the set. The default is the set of all types. For this, as well as other attributes drawn from finite sets, the user is presented with an electric menu for augmenting and removing elements from the set. It is modelled after the electric buffer menu mode, and typing ? when the menu is first presented will display the available commands. The differences between the electric menu and the electric buffer mode are that a space doesn't exit an electric menu, the search commands C-s and C-r are available, and < and > scroll the menu left and right, respectively.

[1] You can access the on-line description of the syntax of regular expressions in EMACS, with M-x info and going to the node (emacs)regexps.

- **creator**—a set of users. A creator passes the filter if it is in the set. The default is the set consisting only of your name; elements are added or deleted from the set in the same way as for types.

- **timestamp**—a pair of strings, each of which is either a timestamp in the usual format or is the empty string (meaning open-ended). A timestamp passes a filter if

  - the first element of the filter is the empty string, or this element is before (inclusive) the timestamp, and
  - the second element of the filter is the empty string, or this element is after (exclusive) the timestamp.

  In other words, the filter defines a time interval, perhaps open at either end. The default is open at both ends.

- **up-to-date**—one of the symbols **+**, **-**, or **+-**; an up-to-date flag passes these filters, respectively, if it indicates the artifact is up-to-date, if it indicates the artifact is out-of-date, or always. The default is **+**.

When you are queried for a filter (either as an argument to **view-plex** or by the **refilter** command), there is always a notion of the "previous filter", perhaps the default. You are prompted to indicate the universal attribute whose filter you wish to change—completion is provided for the names of the attributes—and after doing so are presented with an electric menu indicating the previous value, which you may edit in the minibuffer. Also, you only need to specify an initial segment of a timestamp sufficient to indicate the non-empty element. Typing **C-]** while being prompted for the next universal attribute allows you to avoid going through all the universal attributes once you have changed what you want to change.

A format for the **artifacts** plex is a list of formats for the individual universal attributes, called "column formats". Order matters, because each line in a view on the **artifacts** plex lists the attributes in the order of the column formats. A column format consists of the name of a universal attribute, the width of the column (perhaps 0, which suppresses the column), and some data that depends upon the particular column.

- **name**, **type**, **creator**, **timestamp**—no format data in addition to the width.

- **up-to-date**—two strings, the first to be used when the artifact is up-to-date, the second when it is out-of-date.

As with filters, when you are queried for a format, there is a "previous format". You are prompted for universal attribute names in the order in which they appear in the previous format; to change the order, just type the universal attribute that you want to appear next. Once you indicate which one is next, you are given the chance to change the width, and, if there is additional format data, then that too. The following commands may be useful:

- **C-]**—signals the end of input: use the changes indicated so far, followed by the unchanged columns in their original order.

- **M-]**—restart input with the changes made so far acting as the previous format.
- **M-C-]**—restart the input with the original format.

5.3 The drafts Plex

The **drafts** plex tracks all drafts in the system. A user's principle interaction with it is to provide arguments to commands for drafts (such as those in section 4.3). Filters for this plex are not available, but there is little harm in that since there are never many drafts around at one time.

The format for the **drafts** plex is similar to that for the **artifacts** plex, although a draft has only a subset of the universal attributes of those for an artifact:

- **name**, **type**, **creator**.

(It does not have the **timestamp** or **up-to-date** universal attributes.) In addition to fields for the universal attributes, the **drafts** plex has fields for predecessors and references. If the draft has no predecessors, the corresponding field is blank; if it has one predecessor, the name, type, and timestamp appear; if a draft has more than one predecessor, a number of dashes equal to the number of predecessors appears. A number of asterisks equal to the number of references appears in the references column. [Eventually you will be able to supply an artifact argument by positioning the cursor on a reference in the **drafts** plex.] An entry in the **drafts** plex also has "where" and "initialized" fields. The where column presently has a host and port number, not yet quite reasonable to look at, identifying the E-L session responsible for the draft. If the where column is blank, the draft is said to be *unhosted.* The initialized field is either blank or simply the string "initialized". Drafts may thus be catalogued as follows:

- *system draft*—unhosted and uninitialized
- *abandoned draft*—unhosted and initialized (see section 4.3)
- *starting draft*—hosted and uninitialized
- *normal draft*—hosted and initialized

There are a number of invariants governing the maintenance of the **drafts** plex. The first two of these require the presence of certain lines in this plex.

- For every EMACS buffer on a draft, there is a line in the **drafts** plex whose where field is the host and port number for that buffer's E-L session.
- All referencers of an almost-out-of-date artifact are also almost-out-of-date.

If necessary, the system creates drafts to maintain the latter invariant. We also need an invariant to remove such drafts if edits are aborted, but, before we get to that, it is necessary to understand the roles of the user and the system in editing references. The general principle is that only a person may add or delete a reference, while the system is allowed to change an

existing reference, such as changing a reference to a draft to a reference to a newly committed artifact. Recall that artifacts referenced by an up-to-date artifact have no successor artifacts (by definition). There is a similar rule for references within drafts.

- An artifact that is a reference of a draft must be up-to-date and not almost-out-of-date.

Assume that this holds at the start of an edit. [At present, E-L does not check this.] If an artifact p referenced in a draft d becomes almost-out-of-date during the edit of d, it is because there is some draft d' having p as one of its predecessors; in this case, the system changes the reference of p in d to a reference of d'. Similarly, if d' is subsequently committed, the reference of d' in d is changed to an reference of the new artifact generated by the commit of d'.

A reference in a draft may actually "reference" multiple artifacts or drafts, reflecting support for divergent and convergent evolution and aborting drafts. For example, creating two successors of an artifact (divergent evolution) will create a system draft where the reference corresponding to the now out-of-date artifact will be the two references, one to each successor. Similarly, referencing a draft with two predecessors (convergent evolution) and then aborting the draft will result in a reference (with references) to the two predecessors. Finally, consider the reference of a draft with no predecessors, followed by an abort of the draft. This will result in a reference with a null set of references.

We are finally in a position to describe the elimination of system drafts if edits are aborted.

- A system draft exists only so so long as it has at least one reference that is different from the corresponding reference in its predecessors.

5.4 Scheduling

Whenever you commit an artifact, E-L schedules one or more jobs that take that artifact as input. A job is run by a process called a *job server*. Although job servers may be started and stopped by E-L commands, the lifetime of a job server is independent of any E-L session. Jobs are run one at a time by a given server, but there may be several servers per host and, of course, several hosts on the network. In general, nothing prevents a job server on one host from running jobs scheduled anywhere on the network. We describe the commands for starting and stopping a job server in section 6.1.

The variable `job-done-notice` provides a convenient way to know whether a job that you have scheduled is done. The default value of the variable is the symbol `query`, which means the system will ask you each time you commit an artifact whether you want it to send you a notice when it has completed computing the relevant derivatives. If the variable's value is `nil`, E-L sends no notice, and, if the value is neither `'query` nor `nil`, E-L always sends a notice. Thus, if you want to be notified when a job is done, you can set the value of `job-done-notice` to, say, `t`.

You can also get all the gory details about the status of jobs by viewing the scheduling plexes—`past`, `present`, and `future`.

- **future**—this plex has one line for each job that is scheduled but not yet being run. Columns in this plex are
  - *priority*—a single digit integer indicating the priority of the job; 1 is the highest priority, and 9, the lowest.
  - *start-time*—a timestamp indicating the time at which the job was scheduled.
  - *who*—the person who submitted the job (often by committing an artifact).
  - *tool*—specifies what is to be done. If some kind of information is to be derived, then the job to be run is what is called a deriver for that information (and *tool* will be **derive**); otherwise, it's the name of the specific tool to be run.
  - *args*—specifies arguments for the *tool*. When the *tool* is **derive**, this field is the name and type of the artifact and the kind of information to be derived.
- **present**—has one line for each active job server on the network. Job servers run tools opportunistically. Columns in this plex are
  - *server*—a host and port. If the server is not actively running an E-L tool invocation, the other columns are blank. Otherwise, the other columns give information on a currently running job.
  - *start-time*—carried over from the **future** plex.
  - *who*, *tool*, and *args*—same as in the **future** plex.
- **past**—has one line for each completed tool invocation.
  - *start-time*—carried over from the **present** plex.
  - *stop-time*—time at which the job completed. (The year, month, and day are not displayed.)
  - *who*—carried over from the **present** plex.
  - *results*—in the case of derivers, a string indicating a successful completion (**ok**) or giving some hint of the difficulties encountered.
  - *tool* and *args*—tool-dependent.

In each of these plexes the default order of columns is that given above. [There is presently no meaningful format to change the appearance of these plexes, nor are there meaningful filters for the **future** and **present** plexes, each of which is usually quite short.] A filter for the **past** plex is the same as the filter on the **timestamp** attribute of the **artifacts** plex: a pair, each of which is either a timestamp or the empty string. Only jobs whose *start-time* passes the filter appear in a view on the **past** plex. The default filter has the time of the start of the E-L session as a lower bound and an open-ended upper bound.

The following command offers a convenient way to delete jobs from the future plex:

- **delete-future-jobs**

[There will eventually be a **schedule** plex that provides a way of looking at the **past**, **present**, and **future** plexes in a single window.]

5.5 The notices Plex

The notices buffer discussed in section 2 is actually a view on the **notices** plex. A filter for this plex is a list of users, initialized to the name of the person starting the E-L session (so by default, you see only your own notices). The filter may be changed in the same way as the **creator** attribute of the **artifacts** plex. A format for the **notices** plex consists of three integers: the widths of the name, id, and summary fields.

A view on the **notices** plex is established as part of starting an E-L session, but the window is displayed only if there are notices that pass the filter. Changes to the **notices** plex cause a screen on the buffer to pop up.

If you create several views on the **notices** plex or if you delete all views on it, there are obvious difficulties with the behavior of the default arguments to the **notices** commands described in section 2. Since the notice commands always ask for confirmation, the indeterminacy of default arguments should not be a serious problem.

In the present version of E-L, if you delete all views on **notices**, the system stops alerting you as new notices arrive. The notices are not lost, however. You can see them by viewing the **notices** plex with **C-z C-v**.

5.6 The users Plex

Every user of E-L must be registered in the E-L repository. This happens automatically now, when you start E-L. In the future, installations may want to have some control over the addition and removal of users. With that in mind, we have the following basic EMACS commands:

- **new-user** *name* (this must be a login name)
- **delete-user** *name*

The set of authorized users is kept in a **users** plex, which may be examined in the usual way. You are not permitted to delete a user who is the creator of an artifact. Though you are the default **creator** of artifacts, you can change this default behavior with the command

- **change-active-user** *name*

5.7 The hosts Plex

Every machine that is expected to host a job server must be registered in the repository, for which we have the following commands:

- **new-host** *host machine*
- **delete-host** *host*

where *host* is the local name for an instance of a particular *machine* architecture, like **sun3** or **mips**. Each line in the **hosts** plex simply names a registered *host* and its *machine* type.

6 Troubleshooting

6.1 Servers

The E-L system employs several *servers* in its efforts to support multiple simultaneous users, active views, and opportunistic tool invocation. These include one lock server; one activation server, which invokes the repository checker and automatic deletion mechanism using the information in the `alarm-clock` plex; any number of job servers, entries for each of which appear in the `present` plex; and a pair consisting of a utility server and a message server for each EMACS session. Should it be necessary, you can abort, stop, and start the servers from within E-L with the commands described below or, as a final resort, you can stop them from the shell with `kill -2`.

If you are told that there is no active job server on the network, you may need to start a new one. To do so, use the command

- `start-job-server`

To shut a job server down gracefully, use the following EMACS command:

- `stop-job-server` *server*

The default for a *server* argument is taken from the cursor position if the cursor is in a view on the `present` plex; alternatively, a host and port number, separated by a colon, may be provided. If the server is running a job, shutdown will occur when the job is complete. If the job server has crashed while in the middle of a job, the job data will be moved to the `past` plex. If you want to terminate a job pre-emptively (because, for example, you know it cannot complete), you can issue the command

- `abort-job` *server*

with the cursor on a line in the `present` plex. The job server will try to terminate the specified job and, if successful, will produce a `bad` derivative with an indication of why it was produced. For a stronger hammer, you can prefix the `abort-job` command (with the usual `C-u`); it will then *definitely* abort the job (and may abort other jobs in the same job server as well) and produce a `bad` derivative.

If a job server crashes after it has created an artifact (and updated a view on the `artifacts` plex) but before the artifact has been preseved on disk, the user can see the following warning:

```
Artifact is in a view twice
```

This is only a warning, reflecting an operational problem, not a bug.

Host crashes and other system failures can leave E-L's repository with dangling references to processes—either job servers or E-L sessions—that no longer exist. In such cases, E-L will warn you that it cannot connect to a particular hostport, and it records the warning in a buffer called `*error-log*`. If you believe that the process identified with the hostport is no longer alive, you can safely remove the dangling reference with the command

- **goodbye-hostport** *server*

This will pick up its argument from the cursor's position, much as is done for artifact and draft arguments. For example, you can position the cursor on the appropriate warning message in the ***error-log*** buffer. If the hostport identified an E-L session, then any draft that was hosted by that session will become unhosted (see section 5.3 for a discussion of this aspect of drafts) and can be taken over or deleted (see section 4.3 for a description of **takeover-draft** and **delete-draft**). The command not only excludes the hostport of the user's current session from consideration but also checks, in general, whether the process for a hostport is actually dead.

If the system seems to be taking unduly long to respond, it may be because a server has an exclusive lock on data that you need. You can determine the status of all the locks and servers with the command

- **show-locks** [*server switches*]

You can request information about a particular hostport with the usual notation (host and port separated by a colon) or request some other subset of information with switches:

- -i displays information about the lock server
- -s displays the status of all locking clients
- -p displays the pending lock requests
- -g displays the granted locks
- -c displays the locks held by each client

Giving no server or switches displays the status of all the locks and servers.

6.2 Missing Drafts

If your E-L session crashes, references may exist to drafts that do not appear in the **drafts** plex. In that case, you will be told to use

- **find-missing-draft** *draft-id*

and given the relevant *draft-id* to provide as the argument. The command tries to return a hostport to which you can apply **goodbye-hostport**, after which all references to the draft are gone.

A Customizing LaTeX Output

As we say in section 3.1, each artifact type has its own rules for producing `manuscript` and `caption` derivatives. To encourage uniformity, however, we advertise here several macros that allow users to affect the way a deriver typesets an artifact, its caption, a reference's caption, and any inter-section dividers. By redefining a macro, with `\renewcommand`, in a LaTeX artifact you can change the typesetting of subsequently referenced artifacts. A common practice is to include customizing definitions in the preamble of your `latex-root` referencer. Any deriver using the default definitions should also give reasonable output on most text. Be warned that what we describe below is expected, but not required, behavior of derivers.

- **`\MScaption`** This is used to typeset an artifact's caption (when typesetting the artifact itself). Whatever appears in the caption line—that which follows `Caption:` in the buffer—is passed as the sole argument to this macro. Its default definition

  ```
  \def\MScaption#1{{\rm\bf #1}\hfill\\}
  ```

  typesets the caption line in bold Roman type. If, say, you wanted the caption line in italic type followed by extra vertical space, you could make the following redefinition:

  ```
  \renewcommand{\MScaption}[1]{{\it #1}\hfill\\[2ex]}
  ```

  You can also inhibit the typesetting of the caption altogether by defining the macro to do nothing:

  ```
  \renewcommand{\MScaption}[1]{}
  ```

- **`\MSreference`** This macro is used to typeset a reference's caption, arising when a reference is followed by `<caption>`. By default, it typesets the `caption` derivative in Roman type and encloses it in square brackets, while respecting, of course, any further typesetting commands within the caption line itself. Its default definition is

  ```
  \def\MSreference#1{{\tt [}{\rm #1}{\tt ]}}
  ```

  You could redefine the macro in the following way if you want references to stand out more

  ```
  \renewcommand{\MSreference}[1]{\fbox{\bf #1}}
  ```

- **`\MSuncaptionedreference`** If the caption of a reference is wanted but none has been provided, a `manuscript` deriver would use this macro, with the reference's type as its argument. Its default definition is

  ```
  \def\MSuncaptionedreference#1{Uncaptioned{\tt #1}}
  ```

- **\MSdivider** The editor may display an artifact with dividers to delineate sections of the artifact. A **manuscript** deriver can use **\MSdivider** to typeset such dividers, with the divider's label (e.g. **Targets** for **unix-program** artifacts) as its single argument. The default definition is

```
\def\MSdivider#1{\raisebox{.6ex}{\makebox[\linewidth]{%
 \makebox[1in]{\hrulefill}\raisebox{-.6ex}{\it\ #1 }\hrulefill}}}
```

  which you are free to override.

- **\MSnulldivider** This differs from **\MSdivider** only in that it has no argument for a label. Its default definition is

```
\def\MSnulldivider{\raisebox{.6ex}{\makebox[\linewidth]{%
 \makebox[1in]{\hrulefill}\raisebox{-.6ex}\hrulefill}}}
```

- **\MSsetcaption** This one-argument macro typically appears first in a **manuscript** derivative. It would be passed an artifact's caption, either one explicitly provided by the user or a type-specific default. If neither is available, the command would not appear in the derivative.

```
\def\MSsetcaption#1{\def\MScaptiontext{#1}}
```

  Its function is to make the text of a caption available for use, via **\MScaptiontext**, in other macros.

You would normally put these definitions in a referencing **latex-root** artifact or just before the first reference that you wanted them to affect.

B Automatic Deletion

The system includes a repository checker that looks for inconsistencies in the repository and deletes artifacts according to parameters we describe below. The following observations motivate the deletion policy:

- The time between the creation of artifact p and its youngest successor is a rough measure of the difference in their content and the "stability" of p. If p is succeeded almost immediately, the chances are that the change is small. If p persists for quite a while before it is changed, its content is more likely to be correct. However, as p's age increases it becomes less and less likely to be preferred to its immediate successors.
- If p has many, many successors, it is probably unimportant unless it has somehow been marked as important by a user.

To make precise the policies implied by these observations, we need some definitions. A *successor chain* of length n of artifact p is a sequence of artifacts $p = p_0, p_1, \ldots, p_n$ such that each p_{i-1} is a successor of p_i $(i = 0, \ldots, n-1)$ and p_n has no successors. A *minimal successor chain* for p is one whose length is no greater than any other. Define k_p to be the length of a minimal successor chain. For example, $k_p = 0$ if and only if p has no successors.

Let p' be the youngest successor to p. Let t' be the difference between the timestamp of p' and that of p, and let t_n be the age of p. Define the *age ratio* of p to be $r_p = t_n/t'$. Obviously this definition only makes sense if p has at least one successor.

The criteria for automatic deletion fall into two categories. The *primary deletion criteria* are formulated as conditions under which an artifact may *not* be automatically deleted.

- The artifact has referencers. The reason is obvious.
- The artifact is of type **note**. This allows users to preserve artifacts by including them in **note** artifacts.
- The length of a minimal successor chain is less than two. Thus an artifact must be doubly out-of-date to be considered for automatic deletion.

Once these conditions obtain, deletion depends on the *secondary deletion criteria*, based on two integer parameters:

- R, the maximum allowable age ratio; and
- K, the maximum length of a minimal successor chain.

An artifact p is subject to deletion if it meets the primary deletion criteria and if *either*

$$r_p > R$$

or

$$k_p > K$$

The present value for each of R and K is 20.

To give users an additional last-minute opportunity to retain useful artifacts, we create a list of so-called *doomed artifacts* that meet the deletion criteria; we actually delete the artifacts 24 hours later. You can view this list with the command

- `view-doomed-artifacts`

The command presents the list of doomed artifacts in the format of a default view on the `artifacts` plex. The first two lines show the parameters used to determine the list. Artifact entries may be copied to the minibuffer or used as arguments to commands that take artifact arguments, just as in a view on the `artifacts` plex.

C Installation

This chapter is intended for the installer of the E-L system—someone who wants to set up E-L from the distribution as received via network or tape.

C.1 Prerequisites

An E-L distribution contains:

- E-L sources.
- Sources for Epoch-3.2 E-L, a variant of Epoch-3.2 (patch level 2) that has been modified for use with E-L. Epoch-3.2 is a version of GNU Emacs 18.55 that has been integrated with the X window system.[1]
- Sources for Emacs-18.55-E-L, which is a GNU Emacs that has been customized in the same way as Epoch-3.2-E-L. This version of Emacs is used for batch-mode computations, such as the nightly running of an integrity checker and garbage collector for E-L's repository.

The customizations of these editors made for E-L enable its database views, error summaries, menus, and so on, to be more informative. Furthermore, the customized versions are more reliable than the stock versions from which they derive; they respond more gracefully to signals and transient memory exhaustion, and they can automatically checkpoint certain critical buffers when an interactive session is idle. Each is also suitable as a stand-alone editor, independent of its use for E-L.

Until recently, we did not distribute Emacs as well as Epoch. There is a good deal of common code in the source trees, and much of it (notably the Emacs Lisp library) remains after installation. We have added Emacs to our distribution because of storage corruption bugs that we have fixed in Emacs 18.55; with vanilla Emacs, these bugs sometimes made it impossible to collect repository garbage in the background. Rather than spend time making a merged distribution of Epoch-3.2-E-L and Emacs-18.55-E-L, we intend to move as soon as possible to Epoch 4.2. That system has been structured so that an ordinary, batch-operable Emacs can be produced from the same source tree as Epoch.

In addition to Epoch, E-L uses, or may optionally use, other commonly available systems which are *not* on its distribution tape.

- X windows. E-L is intended to run under X, and it works best on a color monitor. While it can be used acceptably on a monochrome display, E-L has some commands that require color to be useful. For example, a class of "`diff`-based" commands, based on a fancy display of differences between files, artifacts, etc., distinguish old and new versions using color.

[1] Epoch was developed by researchers at the computer science department of the University of Illinois.

- LaTeX. E-L is a highly document-oriented environment, and it depends on the LaTeX typesetting system. If you do not have it already, you'll need to obtain and install LaTeX. (Note: the `makeindex` program distributed with LaTeX is old; a more recent version is available in the `pub/tex/pub` directory at `csc-sun.math.utah.edu`.)

- dvips. E-L also supports `postscript` artifacts, to allow PostScript inserts in LaTeX documents with efficient incremental update processing. The implementation of `postscript` artifacts depends on a particular utility, called `dvips`, for translating LaTeX's output (in "dvi" format) into PostScript. In the US, `dvips` seems to be the most popular and feature-full of the PostScript drivers for dvi files. It is quite well documented and maintained. On the Internet, it can retrieved by anonymous `ftp` from host `labrea.stanford.edu`.

  If you do not plan to use `postscript` artifacts, you don't need to obtain `dvips`, but you will need some means to convert dvi files to printable form.

- Ghostscript. E-L's activity coordination facility requires this PostScript interpreter. It is also the basis for a very good PostScript previewer called `ghostview`. Both can be retrieved from the `pub/ghost` directory at `ftp.cs.wisc.edu`. Version 2.5.2 of Ghostscript works quite well with E-L. A later version is available but has not yet been tried with E-L.

- Lucid Common Lisp. E-L supports programming in Lucid Common Lisp, version 3.0 or later. (The dependence upon Lucid has to do with its foreign function facility and its asynchronous processes. Other Lisps with similar facilities could be similarly supported.) As we will see later in the installation instructions, this is optional on an architecture-by-architecture basis, including the case of no support for Common Lisp at all.

To date, E-L has been ported to Sun3 and Sun4 workstations running SunOS 4.x and to the DecStation 3100 running Ultrix 4.2. All the hosts sharing an installation of E-L must be linked by a common file system.

C.2 Installing E-L

To install E-L, you need three source directory trees, one for E-L (called `e-l-distribution`), one for Epoch (called `epoch-3.2-e-l`), and one for Emacs (called `emacs-18.55-e-l`).

Disk space for installation. The three source trees (E-L, Epoch, Emacs), in uncompressed form, together consume about 30 megabytes of space. If you install E-L for use on N types of machines ("architectures") of which L will have artifact support for Common Lisp, expect the space required to swell to about $28 + 34N + 8L$ megabytes. After everything is in place, you'll need to retain about $18 + 24N + 8L$ megabytes.

To start installation, make an empty directory on in a filesystem with sufficient free space and change into that directory.

Reading sources from tape. If you received E-L on tape, mount the tape and use `tar` to read the tape:

```
$ tar xf /dev/rst0
```

The device `/dev/rst0` used here would be typical for reading a cartridge tape on a Sun workstation. Substitute the correct device file for your tape drive. You will have three subdirectories named `e-l-distribution`, `epoch-3.2-e-l`, and `emacs-18.55-e-l`.

Extracting E-L sources from a compressed tar file. If you received E-L by network file transfer, you can uncompress and extract the E-L source tree from the file `e-l-distribution.tar.Z` as follows.

```
$ zcat e-l-distribution.tar.Z | tar xBf -
```

Once the E-L tree has been extracted, the compressed tar file can be deleted to save space. Repeat the same procedure for the compressed tar files `epoch-3.2-e-l.tar.Z` and `emacs-18.55-e-l.tar.Z`.

Installing Emacs and Epoch Next, install the customized versions of Emacs and Epoch. Since Epoch is a variant of Emacs, the directions for building them are the same. The file `README.E-L` in the `emacs-18.2-e-l` subdirectory just created by `tar` leads to full instructions for building and installing Emacs. The corresponding file in the `epoch-3.2-e-l` directory tells how to install Epoch.

Installing Ghostscript If you plan to use E-L's activity coordination facilities, be sure that your installation of Ghostscript includes support for the PostScript level 2 and Display PostScript extensions. With Ghostscript version 2.5.2, this is done by adding `level2.dev` and `dps.dev` to the definition of `FEATURE_DEVS` in either `unix-cc.mak` or `unix-gcc.mak` (depending on whether you compile Ghostscript with `cc` or `gcc`). The resulting definition typically reads

```
FEATURE_DEVS=filter.dev dps.dev level2.dev
```

Contents of `e-l-distribution`. The file named `MANIFEST` in the `e-l-distribution` subdirectory contains a list of the files and subdirectories in the E-L portion of the distribution. Check to be sure that all seem to have been extracted.

Setting configuration parameters. Set up three directories to hold the permanent results of installation:

- The repository directory, where all the data for E-L is kept. In the example below, the name is `/usr/local/e-l/repository`.
- The bin directory, where E-L programs are kept. The example will use `/usr/local/e-l/bin`.

- The lib directory, where libraries needed for certain extensions of E-L are kept. The example will use `/usr/local/e-l/lib`.

These do not need to be subdirectories of a common directory, as they are in the example. However, they need to be accessible from any host on which E-L will be used. The path leading to each must exist before you install.

Make a copy of `e-l-distribution/configure-e-l.sample`. Call it `configure-e-l`, or whatever you like. We'll refer to it as your configuration script. It records decisions made in setting up E-L, such as where the three permanent directories mentioned above are located.

Following the instructions in `configure-e-l.sample`, edit your configuration script to customize it for your repository. Record the paths of the three permanent directories by editing the values of `CONFIG_REPOSITORY`, `CONFIG_BIN`, and `CONFIG_LIB`.

When E-L refers to programs and other files that are not part of the artifacts system itself, it does so via links that reside in the bin directory. (This ensures that utilities like LaTeX that may be invoked within E-L processes will not be misidentified through some quirk of a particular user's `$PATH` setup.) For each of the remaining `CONFIG_...` variables in the configuration script, supply an absolute path that is appropriate for your system. For each type of host on which you compile E-L, the configuration script will be read and a separate sets of links will be created, so you may use (Bourne shell) conditionals to differentiate among host types if you wish.

In the same directory, the file `universal-paths.el` contains additional absolute paths, those of system header files and commands that may be referred to within C language artifacts. Installation incorporates these paths in `universal` artifacts as a way of encapsulating dependence on the external objects and tracking changes to them. (See the *Language User's Manual.*) Scan the lists for appropriateness at your site; edit and extend them as needed. The set of `universal` artifacts created during initial installation (owned by the pseudo-user `e-l`) will be adjusted accordingly.

Building the E-L executables. All executables are built on site. This minimizes installation difficulties, but it takes a while. To prepare executables for the machine architectures (e.g., `sun3`, `sun4`, or `mips`), on your network, go through the following steps once for each, using a representative host machine.

- Log in to the host, if necessary, and make the `e-l-distribution` directory (created above) your current working directory.
- Using the configuration script that you customized, issue the command

```
$ ./compile-e-l [-no-lisp] ./configuration-script
 output from the compilation process, ending with:
Finished compiling
```

Use the `no-lisp` switch if you do not want lisp support on the particular architecture. The effect of the `compile-e-l` command is to add a subdirectory (named by the machine type) to the `e-l-distribution/bin` directory, and to put the relevant executables into it. It also adds subdirectories to the lib directory. One, named by the machine type, receives object files and archives. Another, called `include`, receives header (`.h`) files.

The installation environment. After the above step, the `e-l-distribution` directory is ready for installation. You must be running X, and Epoch must be available. Be sure your `DISPLAY` environment variable is properly set for your X server, e.g.:

```
$ setenv DISPLAY server-host:0.0
```

E-L uses the environment variables `E_L_REPOSITORY`, `E_L_BIN`, `E_L_LIB`, and `E_L_MACHINE` to communicate with the processes it creates. An attempt to install E-L or to run E-L once installed while you have any of these set in your environment will produce an error message unless your settings agree precisely with the ones E-L means to establish. In short, be sure these variables are not in your environment before installing E-L or starting an E-L session.

The installation program starts a daemon process named `e-l-lock-server` and leaves it running after installation is complete. This lock server process is E-L's resource manager for the whole network of machines sharing E-L, and there should be **only one** instance of it running at any time on the network.[2] Choose as stable and fast a machine as possible to host this resource manager. Log in to that machine (if necessary) to complete the installation, and change your working directory to be the `e-l-distribution` directory.

Installation. Issue the command:

```
$ ./install-e-l ./configuration-script &
```

This starts an Epoch session and displays messages about the progress of the installation. If everything goes normally, Epoch exits when installation is complete. In any case, the progress log is saved in a file called `progress` in the `e-l-distribution` directory. Glance over that file to be sure no errors have occurred.

The next step is to add `e-l` to the command directory (or directories, if there are different file servers for different hosts) from which it will be executable by your E-L user community. A typical choice is `/usr/local/bin/`. You may need root (super-user) privileges do this. Create a symbolic link called `e-l` in the command directory to a shell script of the same name in `/usr/local/e-l/bin/scripts`.[3]

```
$ ln -s /usr/local/e-l/bin/scripts/e-l /usr/local/bin/e-l
```

Finally, you should add `e-l-lock-server` to the set of daemons started at reboot time on the host you have chosen to run the resource manager.[4] Typically, this is done (again, with super-user privileges) by adding the following lines to the end of the script `/etc/rc.local` on that host:

---

[2]That is, one instance per installation of E-L. As will be discussed below under "**Creating a branch repository**", it is sometimes desirable to have multiple instances of the artifacts system at a site, with independent repositories for each. There should be one lock server process (but only one) for each repository.

[3]Because symbolic links are used, however, it is only necessary to have rights to change the command directory when initially installing E-L, or when changing the location of the bin directory, in which case you will need to re-link `e-l` in the obvious way.

[4]While this step is highly desirable to support regular use of E-L, you can omit it if you're prepared to restart the lock server manually whenever its host machine is rebooted. The command to do so requires no special privileges: `e-l -do e-l-start-lock-server`. (The `-do` switch is explained in section C.3.) It is also a very good idea to stop the lock server gracefully before a planned shutdown of its host: `e-l -do e-l-stop-lock-server`.

```
#
# Lock server for E-L
#
if [ -f /usr/local/bin/e-l ]; then
        (/usr/local/bin/e-l -do e-l-start-lock-server)          >/dev/console
fi
```

At this point, E-L is ready to run. Under the C shell, you may need to **rehash** if you have just finished installing E-L, so that the **e-l** command will be recognized. To try it out, start E-L in the background (so as not to tie up your shell window); it will create its own windows:

```
$ rehash
$ e-l &
```

Updating an existing installation. The installation procedure above creates a fresh artifacts repository, completely empty. If the distribution you received is an update for your existing installation, the procedure is slightly different, so that the information in your repository will not be destroyed.

Begin by shutting down all E-L-related processes on your network. To shutdown any existing job servers, use the **M-x stop-job-server** command. Terminate any E-L (Epoch) sessions. Shutdown the resource manager (lock server) using the command

```
$ e-l -do e-l-stop-lock-server
```

Next, compare your old configuration script with the new **configure-e-l.sample**. Merge and edit if necessary to create a new configuration script for your site. Use the same repository, bin, and lib directory paths as in your existing installation. Then follow the instructions in the paragraph above on "Building the E-L executables". Finally, perform the update using the command

```
$ ./update-e-l ./configuration-script &
```

Creating a branch repository. Occasionally, it is useful to create a secondary artifacts repository, one that shares the basic functionality of the primary installation, but that has separate artifacts and plexes, and may even have some different (perhaps experimental) artifact types and derivative classes. We call such a secondary repository a "branch" system—by analogy with a branch office, such as a bank branch. A branch is quick to set up, given the distribution directory and the results of a primary installation, and doesn't use much additional disk space (less than a megabyte, independent of the number of machine architectures accommodated). Use one to test extensions of E-L if you need to be certain of not interfering with production use of the system.[5]

To set up a branch repository, set your working directory to be the distribution directory. If you haven't kept the directory used to perform the original installation on disk, you should

[5]In a future version of E-L, there will be other ways of isolating experimental extensions, so the need for a separate repository will not arise.

recover it and be sure that the configuration script is as you modified it originally. You do *not* need to repeat the `compile-e-l` step, however, because a branch repository shares the compiled C executables and makes copies of the Emacs Lisp and shell-script executables of the original installation.

Now edit the configuration script to define and export the new environment variables `BRANCH_REPOSITORY`, `BRANCH_BIN`, and `BRANCH_LIB` in the same way as for the variables `CONFIG_REPOSITORY`, `CONFIG_BIN`, and `CONFIG_LIB`, but using directory paths chosen for your branch system. For example

```
BRANCH_REPOSITORY="/usr/local/e-l-branch/repository"
BRANCH_BIN="/usr/local/e-l-branch/bin"
BRANCH_LIB="/usr/local/e-l-branch/lib"

export BRANCH_REPOSITORY BRANCH_BIN BRANCH_LIB
```

Again, there is no requirement that these three directories have a common root. You should create each of them before starting branch installation. If any of them is not empty when branch installation begins, you'll be asked whether it is all right to delete the contents.

The command to install a branch E-L is just like that for primary installation, except that `branch-e-l` replaces `install-e-l`:

```
$ ./branch-e-l ./configuration-script &
```

The bin and lib directories share the C-code executables and libraries of the initial installation. However, the subdirectories containing Emacs Lisp code and shell scripts are separate for the branch system, as is the whole repository directory.

Example artifacts. In addition to the artifacts of type `universal` mentioned earlier, the initial installation creates some example artifacts for use in understanding how programs and documents are constructed and interconnected in E-L. These are rooted in an artifact named `Life`, whose creator is the pseudo-user `e-l`. This is the source of a document called *A Literate Implementation in the Artifacts System of the Game of Life.* To view this document on line, you'll need to start a job server (see the next section), then issue the `M-x derive-from` command, giving the artifact `Life` and derivative kind `latex-directory` as arguments. When derivation has completed successfully, use the `M-x preview` and/or `M-x hardcopy` commands to preview or print the document. To try the example program that it describes, issue `derive-from` again, with artifact `Life-C` and kind `Unix-program`. The `M-x execute` and `M-x dbx` commands allow you to run it, once derivation is complete.

C.3 Artifacts System Administration

E-L includes a number of commands for querying and altering the state of associated processes, such as the lock server and the job servers that manage derivations. Among the most useful are `start-job-server`, `stop-job-server`, `show-locks`, `grab-locks`, `start-lock-server`, `stop-lock-server`, and `ping`. Use `M-x describe-function` within E-L to see the on-line documentation of any of these commands.

All of the above commands can also be issued outside E-L, i.e., directly from a shell, using the **-do** argument to **e-l**. For example, the shell-level equivalent of the **M-x show-locks** command with option **-p** would be

```
$ e-l -do e-l-show-locks -p
```

Instead of starting an E-L session, this prints a summary of pending lock requests. In general, the appearance of **-do** as the first argument of **e-l** causes it to set environment variables, including **PATH**, appropriately for starting E-L, but then evaluate its remaining arguments as a shell command line instead of starting a session. The file named **e-l-show-locks** implements the **M-x show-locks** function of E-L, and since **e-l -do** causes it to be found on a search through **$PATH**, the above shell command behaves like the corresponding **M-x show-locks** would in a session.

Incidentally, another very useful **show-locks** option is **-s**, which gives the status of all E-L-related processes (on all hosts) known to the lock server. Information about the lock server itself is obtained with the **-i** option.

C.4 Obtaining Assistance

If you have problems, please contact Glenn Holloway (**glenn@soi.com**) at Software Options (617-497-5054). There is also a mailbox for bug reports about E-L: **e-l-bugs@soi.com**. We encourage you to send problem reports or questions to that address to ensure the fastest response.

Index